

Language Models with GloVe Word Embeddings

Victor Makarek, Lior Rokach, Bracha Shapira
makarek@post.bgu.ac.il, liorrk@bgu.ac.il, bshapira@bgu.ac.il

Department of Software and Information Systems Engineering
Ben-Gurion University of the Negev

1 Introduction

In this work we implement a training of a Language Model (LM), using Recurrent Neural Network (RNN) and GloVe word embeddings, introduced by Pennigton et al. in [1]. The implementation is following the general idea of training RNNs for LM tasks presented in [2], but is rather using Gated Recurrent Unit (GRU) [3] for a memory cell, and not the more commonly used LSTM [4].

2 Motivation

Language Modeling is an important task in many Natural Language Processing (NLP) applications. These application include clustering, information retrieval, machine translation, spelling and grammatical errors correction. In general, a language model defined as a function that puts a probability measure over strings drawn from some vocabulary. In this work we consider a RNN based language model task, which aims at predicting the next n -th word in a sequence, given the previous $n - 1$ words. Put otherwise, finding the word with maximum value for $P(w_n | w_1, \dots, w_{n-1})$. The n parameter is the *ContextWindowSize* argument in the algorithm described further.

To maximize the effectiveness and performance of the model we use word embeddings into a continuous vector space. The model of embedding we use is the GloVe [1] model, with dimensionality size equal to 300 or 50. We use both pre-trained¹ on 42 billion tokens and 1.9 million vocabulary size, and specifically trained for this work vector models, which we trained on SIGIR-2015 and ICTIR-2015 conferences' proceedings.

The model itself is trained as a RNN, with internal GRU for memorizing the prior sequence of words. It was shown lately, that RNNs outperform most language modeling based tasks [2, 3] when tuned and trained correctly.

3 Short Description

In this work we use 300-dimensional and 50-dimensional, GloVe word embeddings. In order to embed the words in a vector space, GloVe model is trained

¹ downloaded from: <http://nlp.stanford.edu/projects/glove/>

by examining word co-occurrence matrix X_{ij} within a huge text corpus. Despite the huge size of the Common Crawl corpus, some words may not exist with the embeddings, so we set these words to random vectors, and use the same embeddings consistently if we encounter the same unseen word again in the text. The RNN is further trained to predict the next word in its embedding form, that is, predicts the next n-dimensional vector, given the *ContextWindowSize* previous words. We divide the *TextFile* into 70% and 30% for training and testing purposes.

4 Pseudo Code

```

input : Input: glove-vectors : Pre-Trained-Word-Embeddings, Text-File,
        ContextWindowSize=10, hidden-unites=300
output: A Language Model trained on Text-File with word-embeddings
        representation
for  $w \in \text{Text-File}$  do
    if  $w \in \text{OOV-file}$  then
        | tokenized-file.append(OOV-file.get-vector(w))
    end
    if  $w \in \text{glove-vectors}$  then
        | tokenized-file.append(glove-vectors.get-vector(w))
    end
    else
        | vector  $\leftarrow$  random-vector()
        | tokenized-file.append(vector)
        | OOV-file.append(vector)
    end
end
NN  $\leftarrow$  CreateSingleHiddenLayerDenseRNN(unit=GRU, inputs=300,
outputs=300, hidden-unites)
NN.setDropout(0.8)
NN.setActivationFunction(Linear)
NN.setLossFunction(MSE)
NN.setOptimizer(rmsprop)
 $X_{train} \leftarrow$  tokenized-file.getInterval(0.0,0.7)
 $X_{test} \leftarrow$  tokenized-file.getInterval(0.7,1.0)
 $Y_{train} \leftarrow X_{train}.\text{Shift}(\text{ContextWindowSize})$ 
 $Y_{test} \leftarrow X_{test}.\text{Shift}(\text{ContextWindowSize})$ 
NN.Fit( $X_{train}, Y_{train}$ )
NN.Predict( $X_{test}, Y_{test}$ )
Algorithm 1: Training a language model on word embeddings

```

5 Detailed Explanation

As stated earlier, GloVe model is trained by examining word co-occurrence matrix of two words i and j : X_{ij} within a huge text corpus. While training

the main idea is stating that $w_i^T w_j + b_i + b_j = \log(X_{ij})$ where w_i and w_j are the trained vectors, b_i and b_j are the scalar bias terms associated with words i and j . The most important parts of the training process in GloVe are: 1) A weighting function f for elimination of very common words (like stop words) which add noise and not overweighted, 2) rare words are not overweighted 3) the co-occurrence strength, when modeled as a distance, should be smoothed with a \log function. Thus, the final loss function for a GloVe model is $J = \sum_{i,j \in V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log(X_{ij}))^2$ where V is a complete vocabulary, and $f(x) = (x/x_{max})^\alpha$ if $x < x_{max}$, and $f(x) = 1$ otherwise. The model that is used in this work was trained with $x_{max} = 100$ and $\alpha = 0.75$.

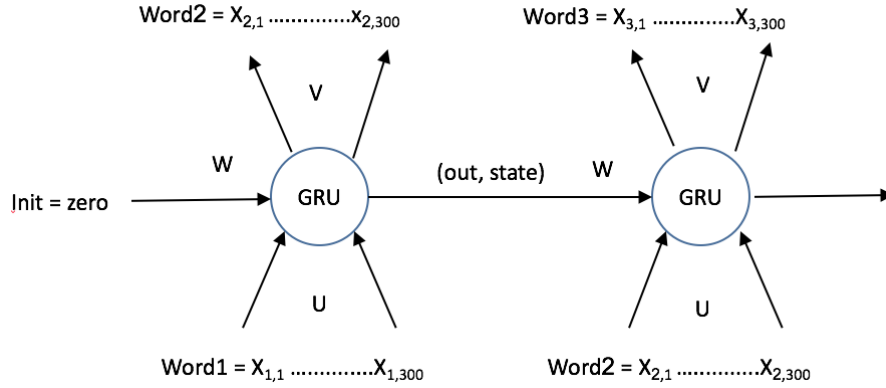


Fig. 1. A general architecture of training an unfolded RNN with 1-sized window based shifted labeled data

Training of the RNN is somewhat blurred between supervised and unsupervised techniques. That is, no extra labeled data is given, but part of the input is used as labels. In this *unfolded* training paradigm, which is illustrated on Figure 1, the output is *shifted* in a way to create a labels for the input train dataset. In this way the RNN can actually learn to predict a next word (vector) in a sequence.

6 Evaluation

6.1 Pre-trained Vector Models

In order to evaluate our implementation of the language model, we train several different language models and compare the predicted error distribution with a random word prediction. On figure 2 we see the error distribution of the RNN with 30 hidden units. The training was performed on 5000 tokens long text file, the first entry at English wikipedia, *Anarchism*.

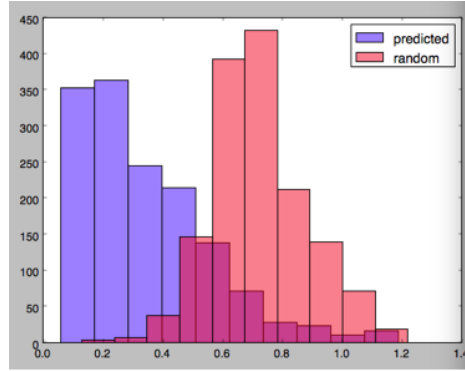


Fig. 2. Two distributions of the predicted next word vector errors. The left is the result of predicted by RNN errors, and the right is predicted by random. The RNN in the model was trained with 30 hidden GRU units. It took 300 iterations (epochs) on the data to achieve these results.

The machine that was used to run the evaluation had the following characteristics: 1.7 GHz, Core i7 with 8 GB memory, OS X version 10.11.13.

The time it took to train the model with 30 epochs was 125 seconds. The time took to make the predictions on a test set is 0.5 seconds.

On figure 3 we see the error distribution of the RNN with 300 hidden units. The time it took to train the model with **300** epochs was 1298 seconds. The time took to make the predictions on a test set is 0.49 seconds.

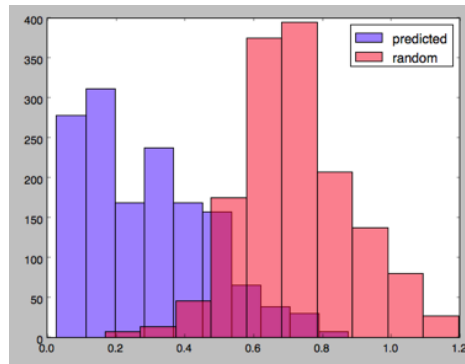


Fig. 3. Two distributions of the predicted next word vector errors. The left is the result of predicted by RNN errors, and the right is predicted by random. The RNN in the model was trained with 300 hidden GRU units. It took 300 iterations (epochs) on the data to achieve these results.

6.2 Self Trained Vector Model

In addition, in order to further evaluate the current approach, we specifically trained a narrow domain-specific, vector model. We used ICTIR-2015 and SIGIR-2015 conferences proceedings as a corpora, and produced 50-dimensional vectors. The *vector model* is based on 1,500,000 tokens total, and resulted in 17,000 long vocabulary. The *language model* for the evaluation was built on a paper published in the ICTIR-2015 proceedings [5]. Consider figure 4. The predicted words' errors distribution differs even more than in the general case, where the vectors were trained on the general Common-Crawl corpora. That is, the performance of the language model, for the task of word prediction is higher.

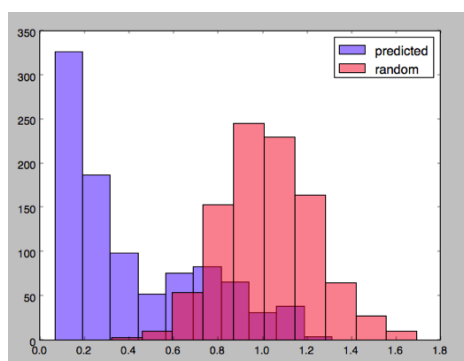


Fig. 4. Two distributions of the predicted next word vector errors. The left is the result of predicted by RNN errors, and the right is predicted by random. The RNN in the model was trained with 50 hidden GRU units. It took 50 iterations (epochs) on the data to achieve these results.

The time took to train this model is 10 seconds. The time took to compute the predictions for evaluations is 0.04 seconds.

7 Instructions for running the code

The implementation of the model training was written in this work in the Python language, version 2.7. The library that was used is Keras, which in the course of this implementation was based on *Theano* framework. Instead of Theano, the Google's *Tensorflow* can be also used behind the scenes of the Keras in this implementation. In order to train the model yourself, you need to follow the next steps:

1. Download pre-trained GloVe vectors from <http://nlp.stanford.edu/projects/glove/>
2. Obtain a text to train the model on. In our example we use a Wikipedia *Anarchism* entry.

3. Open and adjust the `LM_RNN_GloVe.py` file parameters inside the main function:
 - (a) `file_2_tokenize_name` (example = `"/Users/macbook/corpora/text2tokenize.txt"`)
 - (b) `tokenized_file_name` (example = `"/Users/macbook/corpora/tokenized2vectors.txt"`)
 - (c) `glove_vectors_file_name` (example = `"/Users/macbook/corpora/glove.42B.300d.txt"`)
 - (d) `extra_vocab_filename` (example = `"/Users/macbook/corpora/extra_vocab.txt"`).
 This argument has also a default value in the `get_vector` function
4. Run the following methods:
 - (a) `tokenize_file_to_vectors(glove_vectors_file_name, file_2_tokenize_name, tokenized_file_name)`
 - (b) `run_experiment(tokenized_file_name)`

8 Discussion

In this work we implemented and tested the training of a LM based on RNN. To emphasize the strength of such an approach, we have chosen one of the most powerful and prominent techniques for word embeddings - the GloVe embeddings. Although there are other approaches, such as the popular *word2vec* [6] technique, the GloVe embeddings was shown to outperform it on several tasks [1], partially because of the reasons described in section 5. By training the model with two different settings, one of which is order of magnitude more complex than the other we show the power of such LM. The distributions shown on figures 2 and 3 clearly indicate much smaller error on the task of next word prediction.

9 The source at GitHub

The code was submitted publicly to the GitHub repository of the author, and is available under *vicmak* username, *proofseer* project².

² <https://github.com/vicmak/ProofSeer>

References

- [1] Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP). (2014) 1532–1543
- [2] Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. CoRR **abs/1409.2329** (2014)
- [3] Cho, K., van Merriënboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. CoRR **abs/1406.1078** (2014)
- [4] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9** (1997) 1735–1780
- [5] Makarenkov, V., Shapira, B., Rokach, L.: Theoretical categorization of query performance predictors. In: Proceedings of the 2015 International Conference on The Theory of Information Retrieval. ICTIR '15, New York, NY, USA, ACM (2015) 369–372
- [6] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)